Ride Hailing System – Complete Documentation

1. Introduction

This document provides complete documentation for the Ride Hailing System project. The system is designed to demonstrate backend system design, transactional consistency, low-latency APIs, observability, and a minimal frontend for visualizing state transitions.

2. Architecture Overview

The system follows a service-oriented architecture with a NestJS backend, PostgreSQL as the primary datastore, Redis for caching hot paths, and a React frontend. Observability is achieved using New Relic APM.

3. Backend Design

The backend is built using NestJS and follows modular design principles. Core services include Rides, Drivers, Matching, Trips, and Payments. All critical state transitions are protected using database transactions and row-level locks.

4. Data Stores

PostgreSQL acts as the source of truth for all transactional data. Redis is used strictly for caching read-heavy data such as driver location and ride status. Redis is never treated as a source of truth.

5. State Machines

Driver State: AVAILABLE → ON_TRIP → AVAILABLE
Ride State: MATCHING → ASSIGNED
Trip State: CREATED → COMPLETED

6. Frontend Design

The frontend is built using React (Vite) and is intentionally minimal. It uses polling to fetch live updates from the backend and acts as a visual aid for demonstrating backend state changes.

7. API Overview

Key APIs include ride creation, driver location updates, ride acceptance, trip completion, and payment processing. All write APIs support idempotency.

8. Environment Configuration

The backend uses environment variables for database, Redis, and New Relic configuration. The frontend optionally supports environment-based API base URLs.

9. Observability & Performance

New Relic APM is integrated to monitor API latency, throughput, database performance, and errors. Performance testing shows low latency and stable behavior under repeated requests.

10. Conclusion

This project demonstrates a production-minded approach to backend engineering with a focus on correctness, scalability, and observability. The system is easy to reason about, demo, and extend.